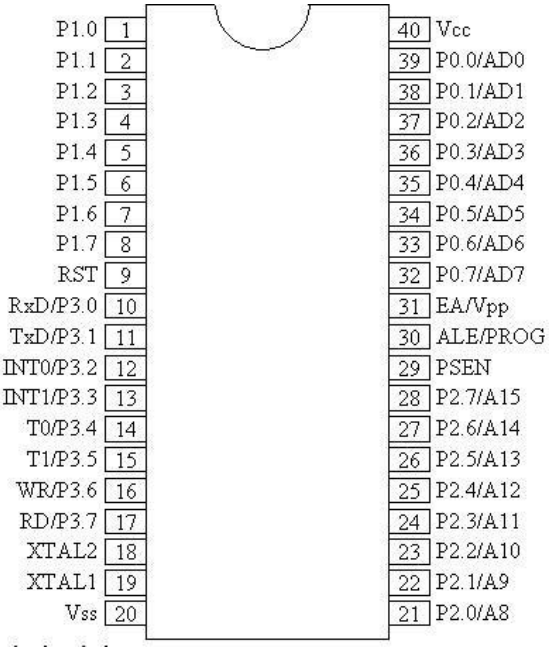
Unit II EMBEDDED HARDWARE AND SOFTWARE

Programming Parallel Port- Serial Port- Timers-Interrupts-Memory- I/O device Interface –Programming Embedded Systems in C- Introduction to Embedded OS- Multitasking OS -RTOS

1. I/O Programming

In 8051, I/O operations are done using four ports and 40 pins. The following pin diagram shows the details of the 40 pins. I/O operation port reserves 32 pins where each port has 8 pins. The other 8 pins are designated as Vcc, GND, XTAL1, XTAL2, RST, EA (bar), ALE/PROG (bar), and PSEN (bar).It is a 40 Pin PDIP (Plastic Dual Inline Package)



I/O Ports and their Functions

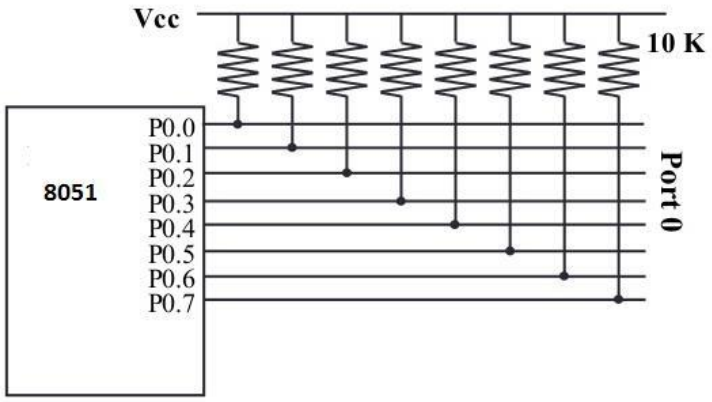
The four ports P0, P1, P2, and P3, each use 8 pins, making them 8-bit ports. Upon RESET, all the ports are configured as inputs, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to a port.

Port 0 (Pin No 32 – Pin No 39)

It has 8 pins (32 to 39). It can be used for input or output. Unlike P1, P2, and P3 ports, we normally connect P0 to 10K-ohm pull-up resistors to use it as an input or output port being an open drain.

It is also designated as AD0-AD7, allowing it to be used as both address and data. In case

of 8031 (i.e. ROMless Chip), when we need to access the external ROM, then P0 will be used for both Address and Data Bus. ALE (Pin no 31) indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE = 1, it has address A0-A7. In case no external memory connection is available, P0 must be connected externally to a 10K-ohm pull-up resistor.




```
MOV A,#0FFH  //(comments: A=FFH(Hexadecimal  i.e. A=1111 1111)
MOV P0,A  //(Port0 have 1's on every pin so that it works as Input)
```

Dual Role of Port 0

Port 0 is also designated as AD0–AD7, as it can be used for both data and address handling. While connecting an 8051 to external memory, Port 0 can provide both address and data. The 8051 microcontroller then multiplexes the input as address or data in order to save pins.

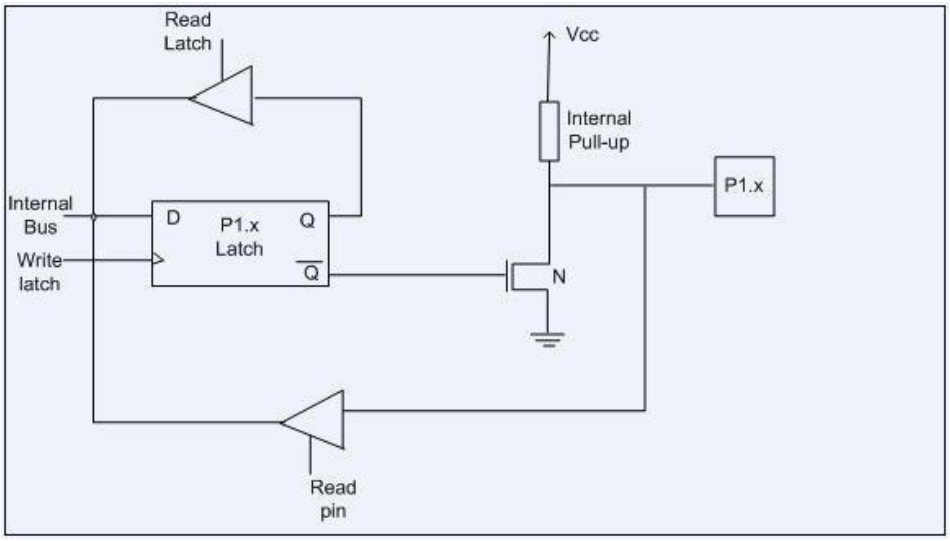
Port 1 (Pin 1 through 8)

It is an 8-bit port (pin 1 through 8) and can be used either as input or output. It doesn't require pull-up resistors because they are already connected internally. Upon reset, Port 1 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to Port 1.

```
Toggle all bits of continuously
        MOV     A,#55
BACK:   MOV     P2,A
        ACALL   DELAY
        CPL     A     //complement(invert) reg. A
        SJMP    BACK
```

If Port 1 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
MOV     A ,#0FFH    //A = FF hex
MOV     P1,A        //Make P1 an input port
MOV     A,P1        //get data from P1
MOV     R7,A        //save it in Reg R7
ACALL   DELAY       //wait
```



Port 2 (Pins 21 through 28)

Port 2 occupies a total of 8 pins (pins 21 through 28) and can be used for both input and output operations. Just as P1 (Port 1), P2 also doesn't require external Pull-up resistors because they are already connected internally. It must be used along with P0 to provide the 16-bit address for the external memory. So it is also designated as (A0–A7), as shown in the pin diagram. When the 8051 is connected to an external memory, it provides path for upper 8-bits of 16-bits address, and it cannot be used as I/O. Upon reset, Port 2 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to port 2.

```
//Toggle all bits of continuously
        MOV     A,#55
BACK:   MOV     P2,A
        ACALL   DELAY
        CPL     A       // complement(invert) reg. A
        SJMP    BACK
```

If Port 2 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
        //Get a byte from P2 and send it to P1
        MOV     A,#0FFH    //A = FF hex
        MOV     P2,A       //make P2 an input port
BACK:   MOV     A,P2       //get data from P2
        MOV     P1,A       //send it to Port 1
        SJMP    BACK       //keep doing that
```

Dual role of Port 2

Besides working as I/O, Port P2 is also used to provide 16-bit address bus for external memory along with Port 0. Port P2 is also designated as (A8– A15), while Port 0 provides the lower 8-bits via A0–A7. In other words, we can say that when an 8051 is connected to an external memory (ROM) which can be maximum up to 64KB and this is possible by 16 bit address bus because we know 216 = 64KB. Port2 is used for the upper 8-bit of the 16 bits address, and it cannot be used for I/O and this is the way any Program code of external ROM is addressed.

Port 3 (Pins 10 through 17)

It is also of 8 bits and can be used as Input/Output. This port provides some extremely important signals. P3.0 and P3.1 are RxD (Receiver) and TxD (Transmitter) respectively and are collectively used for Serial Communication. P3.2 and P3.3 pins are used for external interrupts. P3.4 and P3.5 are used for timers T0 and T1 respectively. P3.6 and P3.7 are Write (WR) and Read (RD) pins. These are active low pins, means they will be active when 0 is given to them and these are used to provide Read and Write operations to External ROM in 8031 based systems.

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

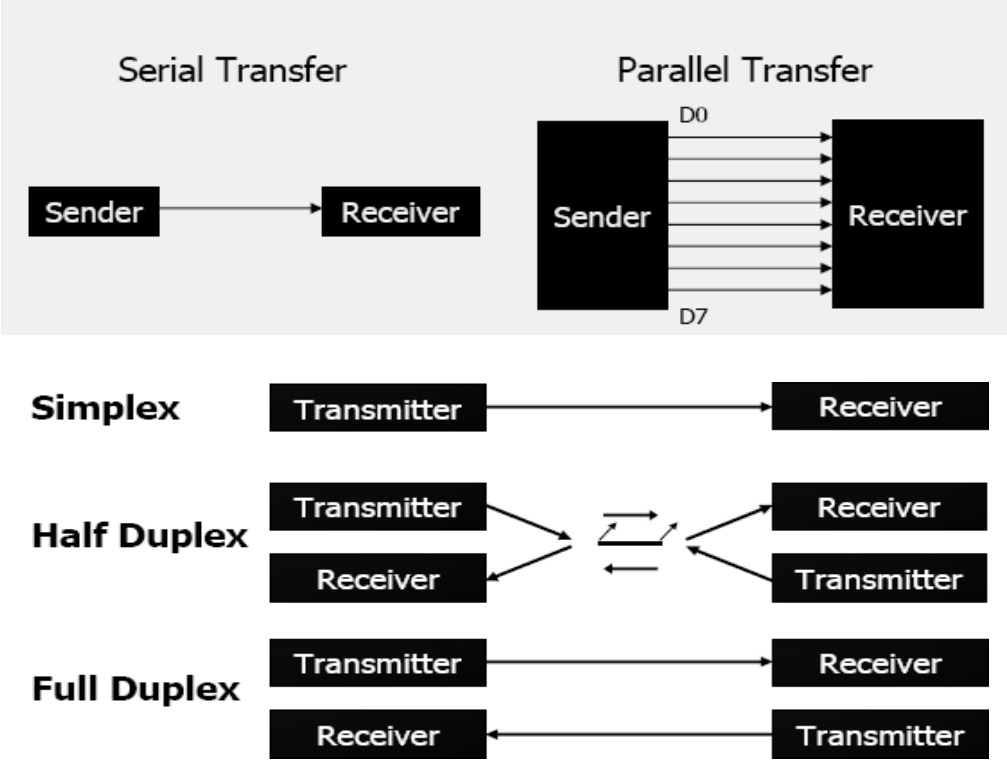
2. 8051 Serial Communication

When a microcontroller communicates with the outside world, it provides the data in byte-sized chunks. In some cases, such as printers, the information is simply taken from the 8-bit data bus and presented to the 8-bit data bus of the printer. This can work only if the cable is not too long, since long cables diminish and even distort signals. Furthermore, an 8-bit data path is expensive. For these reasons, serial communication is used for transferring data between two

systems located at distances of hundreds of feet to millions of miles apart. Fig(1) shows serial versus parallel data transfers. The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication. For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register, then it can be transmitted over a single data line. At the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.

When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation. However, for long-distance data transfers, serial data communication requires a modem to modulate (convert to 0s and 1s to audio tones) and demodulate (converting from audio tones to 0s and 1s).

Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers a block of data(characters) at a time, while the asynchronous method transfers a single byte at a time. There are special IC's made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver transmitter) and USART (universal synchronous asynchronous receiver transmitter).



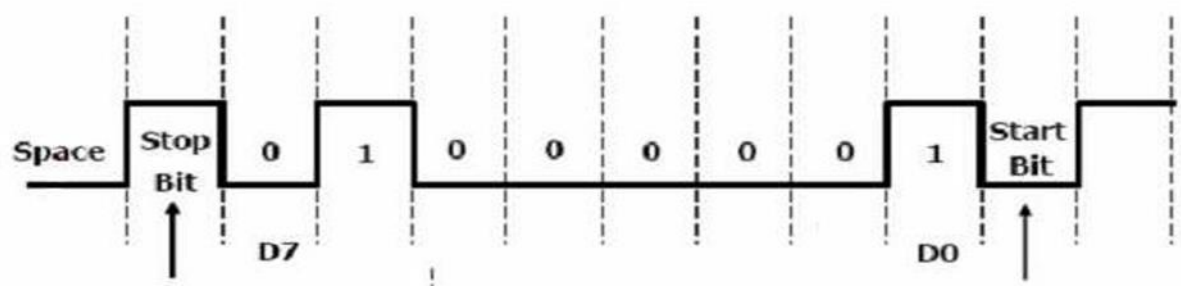
Fig(1)

In data transmission if the data can be transmitted and received, it is a duplex transmission. This is in contrast to simplex transmission such as with printers, in which the computer only sends data. Duplex transmissions can be half or full duplex, depending on whether or not the data transfer can be simultaneous. If data is transmitted one way at a time, it is referred to as half duplex. If the data can go both ways at the same time, it is full duplex. Full duplex requires two wire conductors for the data lines, one for transmission and one for reception, in order to transfer and receive data simultaneously.

Asynchronous Serial Communication.

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s, it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

Asynchronous serial data communication is widely used for character oriented transmissions. In this method, each character is placed between start and stop bits. This is called framing. In data framing for asynchronous communications, the data such as ASCII characters, are packed between a start bit and stop bit. The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit is 1(high). Look at below figure in which the ASCII character "A" (8-bit binary 0100 0001) is framed between the start bit and a stop bit. Notice that the LSB is sent out first.



Figure(2)

Data Transfer Rate

The rate of data transfer in serial data communication is stated in bps (bits per second). Another widely used terminology for bps is baud rate. The baud rate is the modem terminology and s defined as the number of signal changes per second.

8051 SERIAL PORT PROGRAMMING

Baud rate in the 8051

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. This is done with the help of Timer 1. The relationship between thecrystal frequency and the baud rate in the 8051 is shown below.

The 8051 divides the crystal frequency by 12 to get the machine cycle frequency. In the case XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 KHz (11.0592MHz/12 =
921.6 KHz). The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6kHz by 32 once more before it is used by Timer 1 to set the baud rate. Therefore, 921.6 kHz divided by 32 gives 28,800 Hz.

SBUF REGISTER

SBUF is an 8-bit register used solely for serial communication in the 8051. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Similarly, SBUF holds the byte of data when it is received by the RxD line. SBUF can be accessed like any other register in the 8051.

SCON (Serial Control) REGISTER

The SCON register is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things. In this SM0 and SM1 are used to determine the mode. If (SM0,SM1) is (0,0) then it is mode 0, in this mode the serial port function as half duplex serial port with fixed baud rate.If (SM0,SM1) is (0,1) it is mode1,in this mode the serial port function as full duplex serial port with variable baud rate. If (SM0,SM1) is (1,0) then it is mode 2, in this mode the serial port function as full duplex serial port with a baud rate of either 1/32 or 1/64 of the oscillator frequency. If (SM0,SM1) is (1,1) then it is mode 3.The mode-3 is same as mode-2, except the baud rate.In all these modes, only mode 1 is important. When mode 1 is chosen, the following bits are compatible with the COM

port of PCs. They are 8 bits of data with 1 stop bit, and 1 start bit. The mode 1 allows the baud rate to be variable and is set by Timer 1 of the 8051. For each character a total of 10 bits are transferred, where the first bit is the start bit, followed by 8 bits of data, and finally 1 stop bit.

REN is the receive enable. If REN=1, it allows 8051 to receive data on the RxD. If 8051 is to both transfer and receive data, REN must be set to 1. If REN=0, the receiver is disabled and TB8 is used for serial modes 2 and 3. The figure 2 shown below specifies the bits of the SCON register.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication. (Make it 0.)
REN	SCON.4	Set/cleared by software to enable/disable reception.
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

Figure 2 SCON register

Programming the 8051 to transfer data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 to set the baud rate.
2. The TH1 is loaded with one of the values in Table (2) to set the baud rate for serial datatransfer.
3. The SCON register is loaded wit the value 50H, indicating serial mode 1, where an 8-bitdata is framed with start and stop bits.
4. TR1 is set to 1 to start Timer 1.
5. TI is cleared by the “CLR TI” instruction .
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction “JNB TI, xx” to see

if thecharacter has been transferred completely.

8. To transfer the next character, go to Step 5.

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV   TMOD,#20H   ;timer 1,mode 2(auto reload)
        MOV   TH1,#-6     ;4800 baud rate
        MOV   SCON,#50H   ;8-bit, 1 stop, REN enabled
        SETB  TR1         ;start timer 1
AGAIN:  MOV   SBUF,#"A"   ;letter "A" to transfer
HERE:   JNB   TI,HERE     ;wait for the last bit
        CLR   TI          ;clear TI for next char
        SJMP  AGAIN       ;keep sending A
```

Programming the 8051 to receive data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode2 to set the baud rate.
2. TH1 is loaded with one of the values in Table(2) to set the baud rate.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bitdata is framed with start and stop bits and receive enable is turned on.
4. TR1 is set to 1 to start Timer 1.
5. RI is cleared with the "CLR RI" instruction.
6. The RI flag bit is monitored with the use of the instruction "JNB RI, xx" to see if anentire character has been received yet.
7. When RI is raised, SBUF has the byte. Its contents are moved into safe place.
8. To receive the next character, go to Step 5.

Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit

Solution:

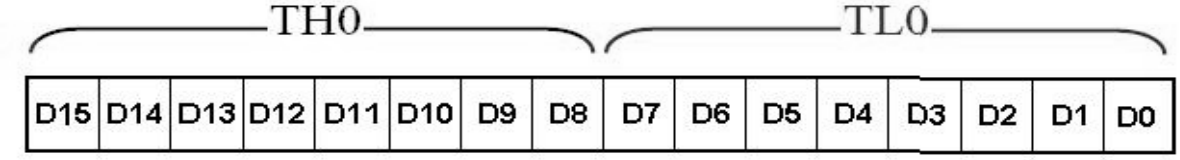
```
        MOV   TMOD,#20H   ;timer 1,mode 2(auto reload)
        MOV   TH1,#-6     ;4800 baud rate
        MOV   SCON,#50H   ;8-bit, 1 stop, REN enabled
        SETB  TR1         ;start timer 1
HERE:   JNB   RI,HERE     ;wait for char to come in
        MOV   A,SBUF      ;saving incoming byte in A
        MOV   P1,A        ;send to port 1
        CLR   RI          ;get ready to receive next
                          ;byte
        SJMP  HERE        ;keep getting data
```

3. PROGRAMMING 8051 TIMERS

The 8051 has two timers; timer 0, timer 1. They can be used either as timers or as event counters. Both timer 0 and timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture , each 16 bit timer is accessed as two separate registers of low byte and high byte.

TIMER 0 registers

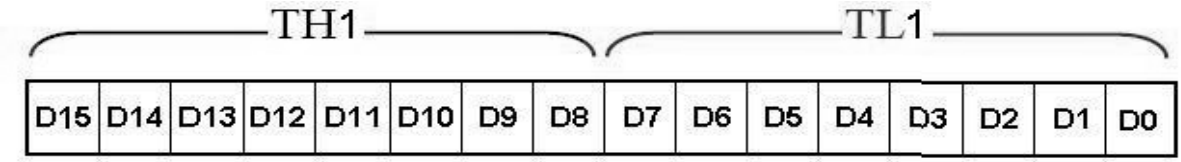
The 16 bit register of timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (timer 0 low byte) and the high byte register is referred to as TH0 (timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2 etc. For example, the instruction " MOV TL0,#25H" loads the value 25H into TL0.



Fig(1): Timer 0 Registers

TIMER 1 registers

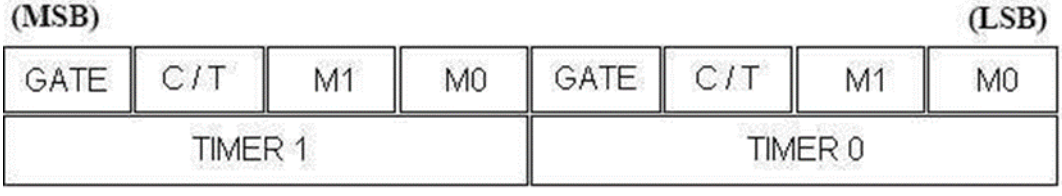
Timer 1 is also 16 bits, and its 16 bit register is split into two bytes, referred to as TL1(timer 1 low byte) and TH1 (timer 1 high byte). These registers are accessible in the same way as the registers of timer 0.



F ig(2): Timer 1 Registers

TMOD (Timer Mode) Register

Both timers 0 and 1 use the same register, called TMOD, to set th various timer operation modes. TMOD is an 8-bit register in which the lower 4 bits are set aside for timer 0 and the upper4 bits are set aside for timer 1. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation. TMOD register is shown in fig(3).



Fig(3): TMOD Register

GATE: The T MOD register of Fig(3) that both timers 0 and 1 have the GATE bit. Every timerhas means of starting and stopping. Some timers do this by software, some by hardware, and some both software and hardware controls. The timers in the 8051 have both. The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1. This is achieved by the instructions "SETB TR1" and "CLR T R1" for timer 1 and "SETB TR0" and "CLR TR0" for time 0. The SETB instruction starts it, and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE=0 in the TMOD register.

M1, M0: M0 and M1 select the timer mode. As show in the below Table, there are three modes; 0, 1, and 2. Mode 0 is a 13 bit timer, mode 1 is a 16 bit timer and mode 2 is an 8-bit timer.

M1	M2	MODE
0	0	0
0	1	1
1	0	2
1	1	3

C / T (Clock / Timer): This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter. If C/T =0, it is used as a timer for time delay generation. The clock source for the time delay is the crystal frequency of the 8051.

Timer Programming

Mode 1 Programming

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer, therefore it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high flag bit called TF (timer flag). This timer flag can be monitered. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TR0" for Timer 0 and "CLR TR1" for Timer 1. Notice that each timer has its own timer flag: TF0 for Timer 0 and TF1 for Timer 1.
4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to '0'.

Steps to Program in Mode 1

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count values
3. Start the timer.
4. Keep monitoring the timer flag (TF). Get out of the loop when TF becomes high
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.

Example

	MOV TMOD, #01	Time 0, mode 1 (16-bit mode)
HERE:	MOV TL0, #F2H	TL0 = F2H, the Low byte
	MOV TH0, #FFH	TH0 = FFH, the High byte
	CPL P1.5	
	ACALL DELAY	
	SJMP HERE	

Delay using Timer 0

DELAY:

	SETB TR0	Start Timer 0
AGAIN:	JNB TF0, AGAIN	Monitor Timer 0 flag until ir rolls over
	CLR TR0	Stop Timer 0
	CLR TF0	Clear Timer 0 flag
	RET	

Mode 2 Programming

The following are the characteristics and operations of mode 2:

1. It is an 8-bit timer, therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (Timer Flag). If we are using Timer 0, TF0 goes high. If we are using Timer 1, TF1 is raised.
4. When the TL register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL

   It must be emphasized that mode 2 is an 8-bit timer. However, it has an auto-reloading capability. In auto-reload, TH is loaded with the initial count and a copy of it is given to TL. This reloading leaves TH unchanged, still holding a copy of the original value. This mode has many applications, including setting the baud rate in serial communication.

Steps to program in mode 2

To generate the time delay using the timer's mode 2, take the following steps.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used, and select the timer mode (mode 2)
2. Load the TH registers with the initial count value
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "JNB TF0, Target" or "JNB TF1, Target" instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to step 4, since mode 2 is auto-reload.

Program: Assembly Language Program to generate the square on pin P1.0, assuming XTAL = 11.0592 MHz.

```
        MOV TMOD, #20H ; T1/ mode 2/ 8-bit/ auto-reload
        MOV  TH1, #05H     ; TH1 = 05
        SETB TR1           ; Start Timer 1
BACK:   JNB  TF1, BACK     ; stay until timer rolls over
        CPL  P1.0          ; complement P1.0 to get high, low
        CLR  TF1           ; clear Timer 1 flag
        SJMP BACK          ; mode 2 is auto-reload
```

COUNTER PROGRAMMING

The timer / counter of the 8051 is used to generate time delays. These timers can also be used as counters counting events happening outside the 8051. As far as the use of a timer as an event counter is concerned, everything that in programming the timer applies to programming it as a counter, except tha source of the frequency. When the timer/counter is used as a timer, the 8051's crystal is used as the source of the frequency. When it is used as a counter, it is a pulse outside the 8051 that increments the TH, TL register.

C/T bit in TMOD register

The C/T bit in the TMOD register decides the source of the clock for the timer. If C/T=0, the timer gets pulses from the crystal. In contrast, when C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051. Therefore, when C/T = 1, the counter counters up as pulses are fed from pins 14 and 15. These pins are called T0 (Timer 0 input) and T1 (Timer 1 input). These two pins belong to port 3. In the case of Timer 0, when C/T = 1, pin P3.4 provides the clock pulse and the counter counts up for each clock pulse coming from that pin. Similarly, for Timer 1, when C/T = 1 each clock pulse coming in from pin P3.5 makes the counter count up.

Port 3 Pins Used For Timer 0 and Timer 1

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer / Counter 0 external input
15	P3.5	T1	Timer / Counter 1 external input

(MSB) (LSB)

GATE	C/T	M1	M2	GATE	C/T	M1	M2
Timer 1				Timer 0			

Fig:TMOD register

TCON REGISTER

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig : TCON register

TCON register is an 8-bit register. As shown in above figure the upper four bits are used to store the TF and TR bits of both Timer 0 and Timer 1. The lower four bits are set aside for controlling the interrupt bits. TCON is a bit addressable register.

4.INTERRUPTS

An interrupt is an internal or external event that interrupts the microcontroller to inform it that a device needs its service. Whenever any device needs its service, the device notifies the microcontroller by sending it as interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program which is associated with the interrupt is called interrupt Service Routine (ISR). The microcontroller can serve many devices based on the priority assigned to it.

Execution of an Interrupt

In order to use any interrupt, the following steps must be taken.

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.

2. It also saves the current status of all the interrupts internally.

3. It jumps to a fixed location in memory called the interrupt vector or table that holds the address of the Interrupt Service Routine (ISR).

4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RET 1.

5. Upon executing RET 1 instruction, the microcontroller returns to the place where it was interrupted. First it gets the program counter (PC) address from the stack by popping the top two byes of the stack into the PC. Thenit startsto execute from that address.

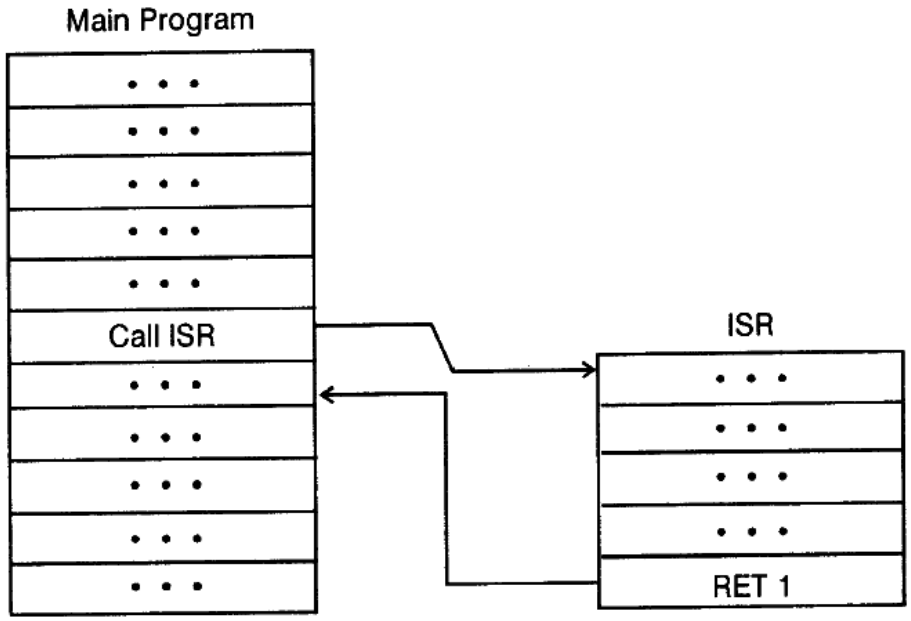


Figure 4:Interrupt Structure

Interrupts in 8051

- Five interrupts are provided in the 805l.
- Three of these a regenerated by internal operations: Timer Flag 1 & 0, and the serial port interrupt (RI or TI).
- Two interrupts are triggered by external signals provided by circuitry that is connected to pin

INT0 and **INT1** (port pins P3.2 and P3.3)

Types		**Interrupt**	**Vector Address**
Internal	TF0	Timer flag 0 interrupt	$000B_H$
	TF1	Timer flag 1 interrupt	$001B_H$
	RI/TI	Serial port interrupt	0023_H
External	$\overline{\text{INT0}}$	External interrupt 0	0003_H
	$\overline{\text{INT1}}$	External interrupt 1	0013_H

Table 3: Interrupt Vector

a) Timer flag interrupts

• When a timer / counter overflows, the corresponding timer flag TF0 or TFI (location: 000B H or 001B H) is set to l.

• The flag is cleared to 0 when the resulting interrupt generates a program call to the appropriate timer subroutine in memory.

b) External interrupts

• The external hardware interrupts INT0 andINT1 are located on pins P3.2 and P3.3.

• Inputs on these pins can set the interrupt flags IE0 and IE1 in the TCON register to 1 by level triggering or edge-triggering.

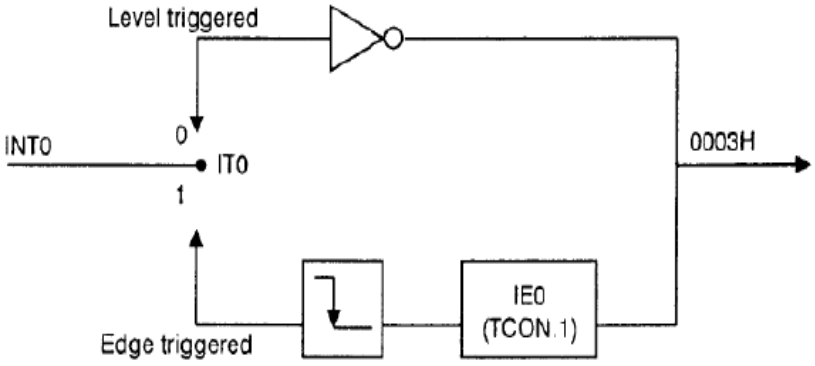


Figure 5: Activation of INT0

• Fig. 4. Shows the activation of INT0 and Fig.5. shows the activation of INT1

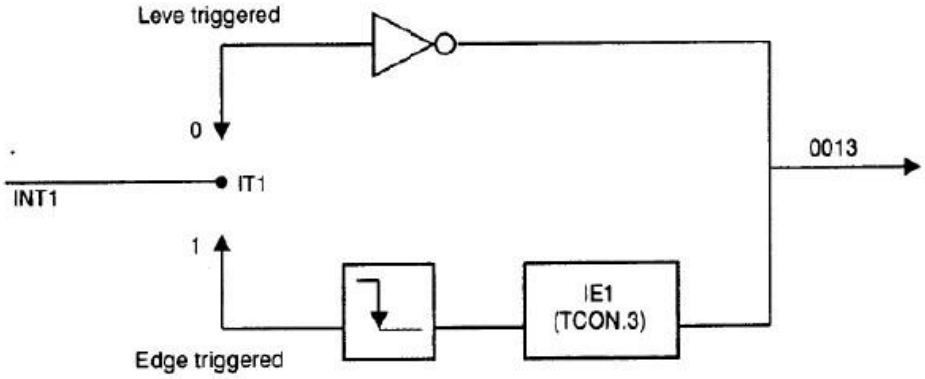


Figure 6: Activation of INT1

c) Serial Port Interrupt

- In SCON, if RI = l, a data byte is received If TI = 1, a data byte has been transmitted.
- These are ORed together to provide a single interrupt to the processor.
- The interrupt bit in the IE register is used to both send and receive data.
- If IE.4 [ES- Enable serial port interrupt] is enabled, when RI or TI is raised and 8051gets interrupted and jumps to memory address location 0023H to execute the ISR.
- The Fig.6.Showsthe serial interrupt is invoke by TI or RI flags.

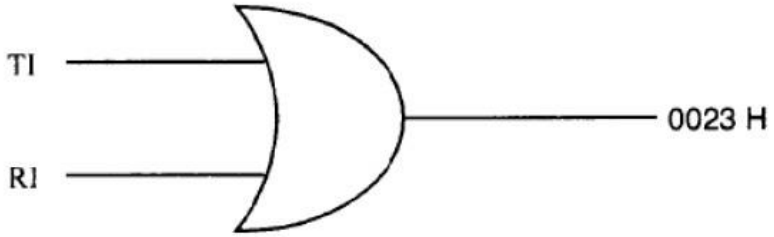


Figure 7: Serial Port Interrupt

INTERRUPTCONTROL

All interrupt functions are under the control of the program. The programmer is able to alter control bits in the:

- Interrupt Enable Register (IE)
- Interrupt Priority Register (IP) and
- Timer Control Register (TCON).

Interrupt Enable Register (IE)

- The IE register holds the programmable bits that can enable or disable all the interrupts.
- Bit D7 of the IE register (EA) must be set high to allow the rest of the register to take effect.
- If EA = l, interrupts are enabled and will be responded to if their corresponding bits in IE are high.
- If EA = 0, no interrupt will be responded to, even if the associated bit in the EI register is high.

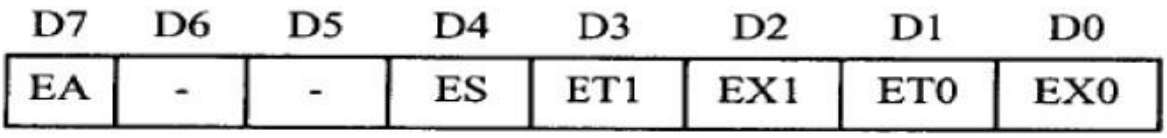


Figure 8: IE Register

EA: Enable interrupts bits.

- Set to 1 to permit individual interrupts to be enabled by their enable bits.
- Cleared to 0 by program to disable all interrupts.

ES: Enable serial port interrupt.

- Set to 1 to enable by program.
- Cleared to 0 to disable serial port interrupt.

ET1: Enable/ disable the Timer 1 overflow interrupt.

EX1: Enable external interrupt 1.

- Set to 1 by program to enable**INT1'**interrupt.
- Clearedto 0 to disable**INT1**'interrupt.

ET0: Enable / disable the Timer 0 overflow interrupt.

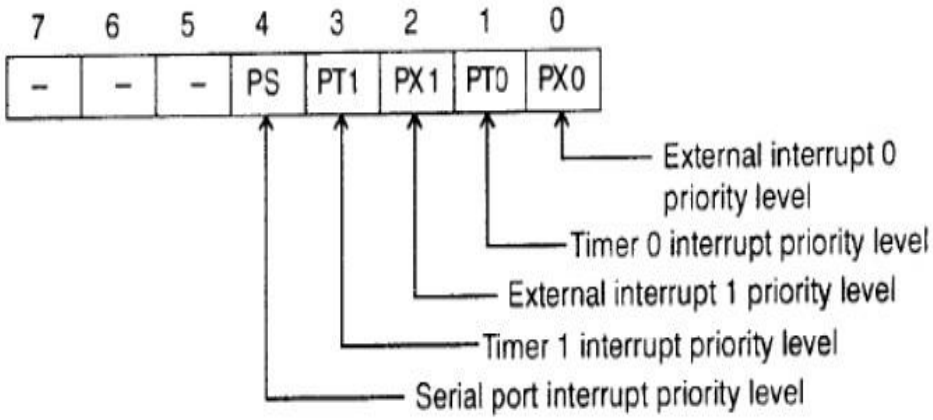
EX0: Enable/ disable the external interrupt 0.

- Set to 1 by program to enable **INT0'** interrupt.
- Clearedto 0 to disable**INT0**' interrupt.

Interrupt Priority Register (IP)

- Interrupt priority (IP) register determines the interrupt priority.
- Bits in IP registers set to 1 give the accompanying interrupt a high priority; a 0 assigns a low priority.
- Interrupts with a high priority can interrupt another interrupt with a lower priority and the lower priority continues after the higher is finished.
- If two interrupts with the same priority occur at the same time, then they have the following ranking:

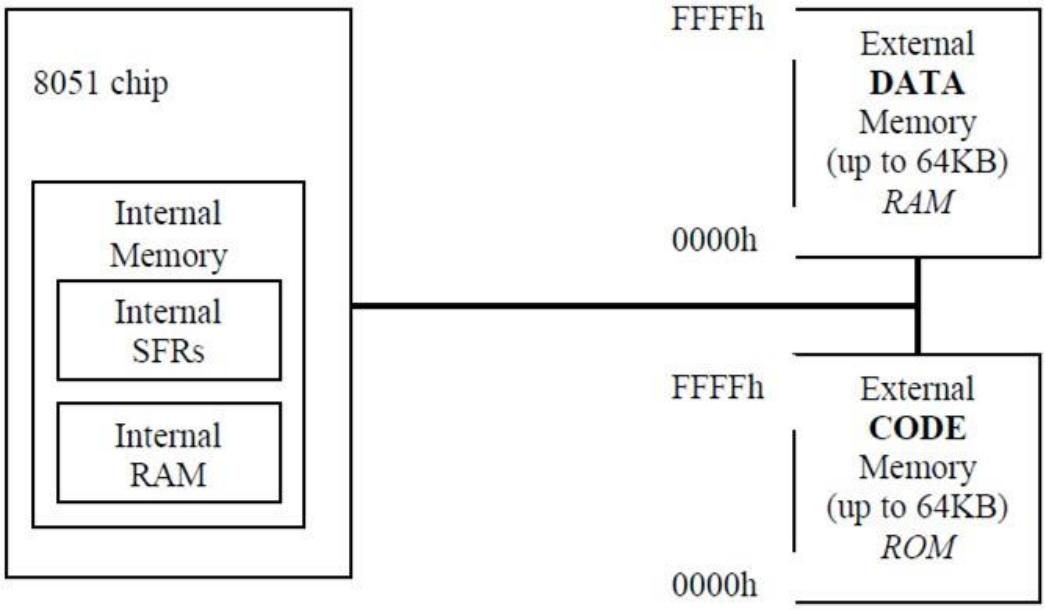
1. IE0
2. TF0
3. IE1
4. TF1
5. RI/TI



- The bit addressable IP register is shown in Fig.9. If the bit is 0, the corresponding interrupts has a lower priority, otherwise higher priority.

MEMORY ORGANISATION:

- The 8051 has a separate memory space for programs (Code) and data.
- Program memory stores the programs to be executed, while data memory stores the data like intermediate results, variable and constants required for the execution of the program.



External Program (Code) Memory

- The executable program is stored in this code memory.
- The code memory size is limited to 64KBytes (in a standard 8051).
- The code memory is read-only in normal operation and is programmed under special conditions. e.g. it is a PROM or a Flash RAM type of memory.

External RAM Data Memory

- This is read-write memory and is available for storage of data. Up to 64KBytes of external RAM data memory is supported (in a standard 8051).

Internal Memory

The 8051's on-chip memory consists of 256 memory bytes organized as follows:

The first 128 bytes of internal memory is organized as shown in figure 2, and is referred to as Internal

RAM, or IRAM.

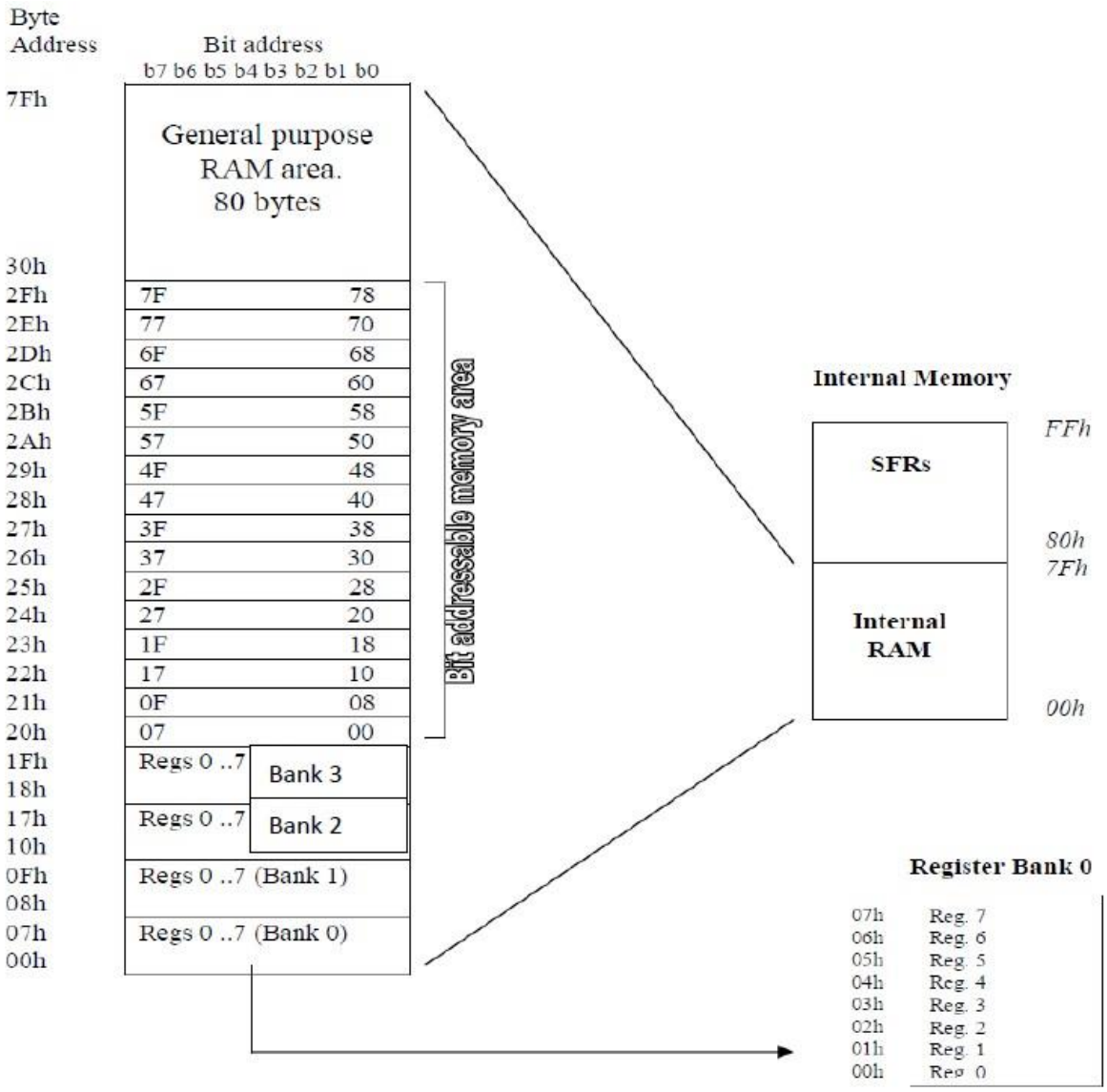


Figure : Organization of Internal RAM (IRAM) memory

Register Banks: 00h to 1Fh

The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5, R6, and R7).

These registers are used in instructions such as:

ADD A, R2 ; adds the value contained in R2 to the accumulator

Bit Addressable RAM: 20h to 2Fh

- The 8051 supports a special feature which allows access to bit variables.
- This is where individual memory bits in Internal RAM can be set or cleared.

- In all there are 128 bits numbered 00h to 7Fh.
- Being bit variables any one variable can have a value 0 or 1.
- A bit variable can be set with a command such as SETB and cleared with a command such as CLR.

General Purpose RAM: 30h to 7Fh

- These 80 bytes of Internal RAM memory are available for general-purpose data storage.
- Access to this area of memory is fast compared to access to the main memory and special instructions with single byte operands are used.
- However, these 80 bytes are used by the system stack and in practice little space is left for general storage.
- The general purpose RAM can be accessed using direct or indirect addressing modes.

SFR Registers

- The SFR registers are located within the Internal Memory in the address range 80h to FFh, as shown in figure 3.
- Each SFR has a very specific function.
- Each SFR has an address (within the range 80h to FFh) and a name which reflects the purpose of the SFR.
- Although 128 byes of the SFR address space is defined only 21 SFR registers are defined in the standard 8051.
- Undefined SFR addresses should not be accessed as this might lead to some unpredictable results.
- Note some of the SFR registers are bit addressable. SFRs are accessed just like normal Internal RAM locations.

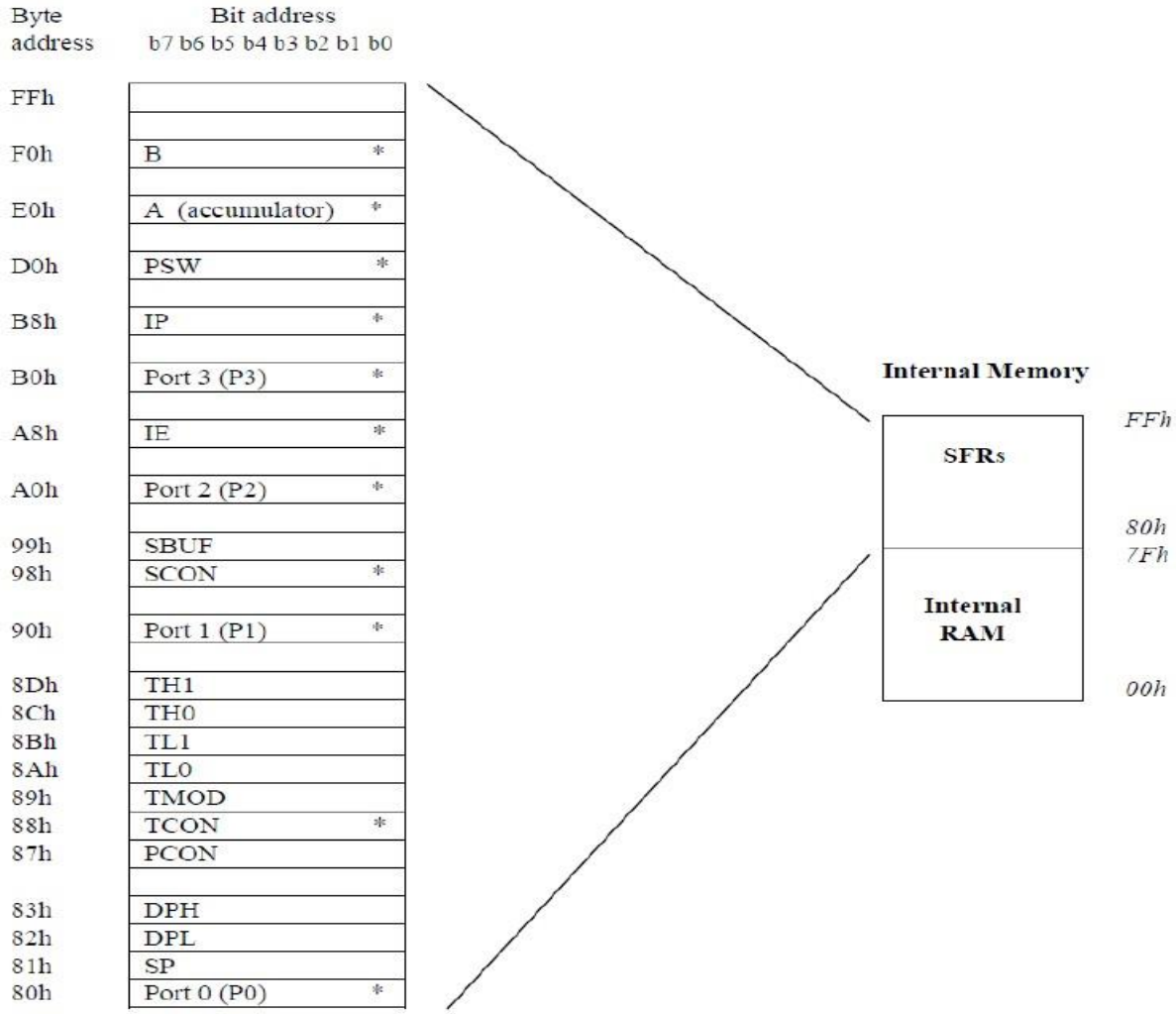


Figure. SFR Register Layouts

5. Interfacing a Microprocessor to Keyboard

The key board interfaced is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8matrix key board. So only two ports of 8051 can be easily connected to the rows andcolumns of the key board.

Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high which indicates microcontroller that the key is pressed. By this high on

the bit key in the corresponding column is identified.

Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.

To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out.

Once we get the row next job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The contentof the input port is rotated with carry until the carry bit is set.

The contents of the counter are then compared and displayed in the display. This displayis designed using a seven segment display and a BCD to seven segment decoder IC 7447.The BCD equivalent number of counter is sent through output part of 8051 displays thenumber of pressed key.

Fig 5.1 Interfacing Keyboard to 8051 Microcontroller

5.1 Interfacing Analog to Digital Data Converters

- In most of the cases, the PPI 8255 is used for interfacing the analog to digital converters with microprocessor.
- The analog to digital converters is treaded as an input device by the microprocessor, that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output ofthe ADC are carried out by the CPU using 8255 I/O ports.
  - The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
  - It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
  - The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
  - General algorithm for ADC interfacing contains the following steps:
    1. Ensure the stability of analog input, applied to the ADC.
    2. Issue start of conversion pulse to ADC
    3. Read end of conversion signal to mark the end of conversion processes.
    4. Read digital data output of the ADC as equivalent digital output.
    5. Analog input voltage must be constant at the input of the ADC right from the start of Conversion to of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
    6. If the applied input changes before the complete conversion process is over,

the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809:

The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100μs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.

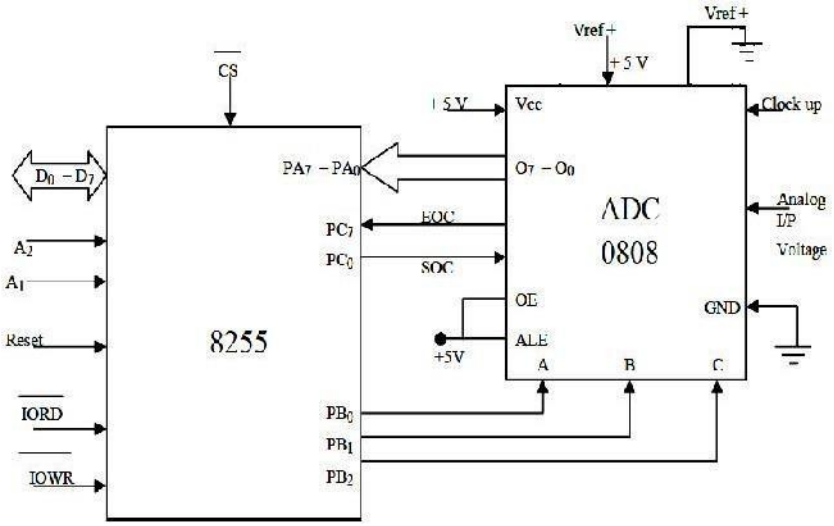


Fig.5.2 Interfacing ADC with 8255 and Microcontroller

These converters internally have a 3:8 analog multiplexers so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

There is unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do not contain any internal sample and hold circuit. If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

5.2 External Memory Interface:

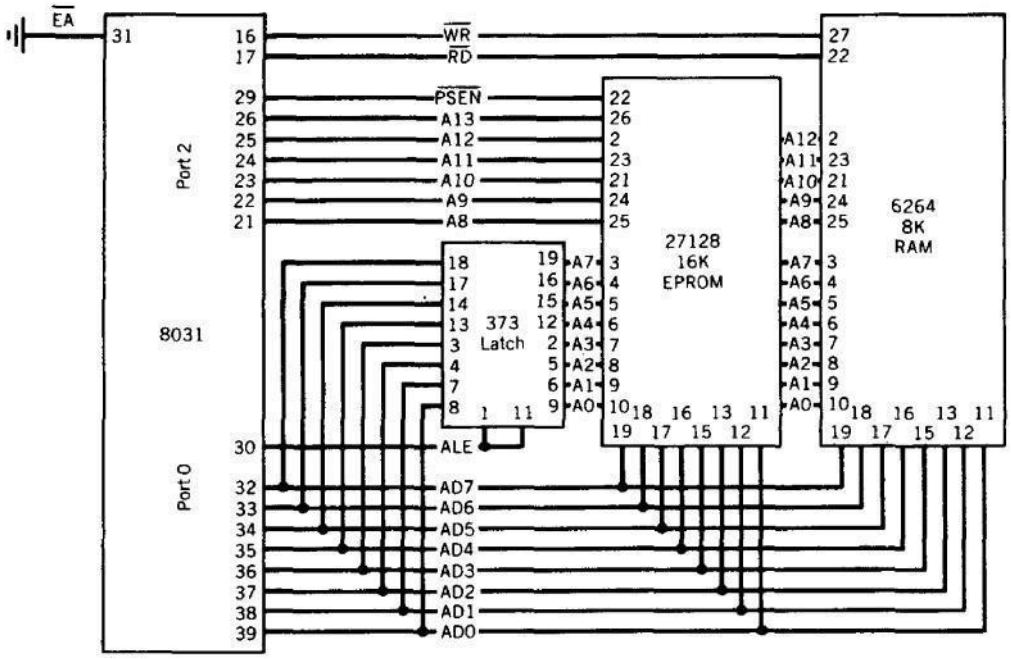


Fig.5.3 Interfacing External Memories with Microcontroller

8031 chip is a ROM less version of the 8051. In other words, it is exactly like any member of the 8051 family such as the 8751 or 89C51 as far as executing the instructions and features are concerned, but it has no on-chip ROM. Therefore, to make the 8031 execute 8051 code, it must be connected to external ROM memory containing the program code. In this section we look at interfacing the 8031 microcontroller with external ROM. Before we discuss this topic, one might wonder why someone would want to use the 8031 when they could buy an 8751, 89C51, or DS5000. The reason is that all these chips have a limited amount of on-chip ROM. Therefore, in many systems where the on-chip ROM of the 8051 is not sufficient, the use of an 8031 is ideal since it allows the program size to be as large as 64K bytes. Although the 8031 chip itself is much cheaper than other family members, an 8031-based system is much more expensive since the ROM containing the program code is connected externally and requires more supporting circuitry, as we explain next. First, we review some of the pins of the 8031/51 used in external memory interfacing.

EA pin

In 8751/89C51/DS5000-based systems, we connect the EA pin to V_{cc} to indicate that the program code is stored in the microcontroller's on-chip ROM. To indicate that the program code is stored in external ROM, this pin must be connected to GND. This is the case for the 8051-based system. In fact, there are times when, due to repeated burning and erasing of on-chip ROM, its UV-EPROM is no longer working. In such cases one can also use the 8751 (or 89C51 or any 8051) as the 8031. All we have to do is to connect the EA pin to ground and connect the chip to external ROM containing the program code.

5.3 Stepper Motor Interface

The complete board consists of transformer, control circuit, keypad and stepper motor as shown in snap.

The circuit has inbuilt 5 V power supply so when it is connected with transformer it will give the supply to circuit and motor both. The 8 Key keypad is connected with circuit through which user can give the command to control stepper motor. The control circuitincludes micro controller 89C51, indicating LEDs, and current driver chip ULN2003A. One can program the controller to control the operation of stepper motor. He can give different commands through keypad like, run clockwise, run anticlockwise, increase/decrease RPM, increase/decrease revolutions, stop motor, change the mode, etc.

Unipolar stepper motor: - unipolar stepper motor has four coils. One end of each coil is tied together and it gives common terminal which is always connected with positive terminal of supply. The other ends of each coil are given for interface.

By means of controlling a stepper motor operation we can

1. Increase or decrease the RPM (speed) of it
2. Increase or decrease number of revolutions of it
3. Change its direction means rotate it clockwise or anticlockwise

To vary the RPM of motor we have to vary the PRF (Pulse Repetition Frequency). Number of applied pulses will vary number of rotations and last to changedirection we have to change pulse sequence.

So all these three things just depend on applied pulses. Now there are three different modes to rotate this motor

1. Single coil excitation
2. Double coil excitation
3. Half step excitation

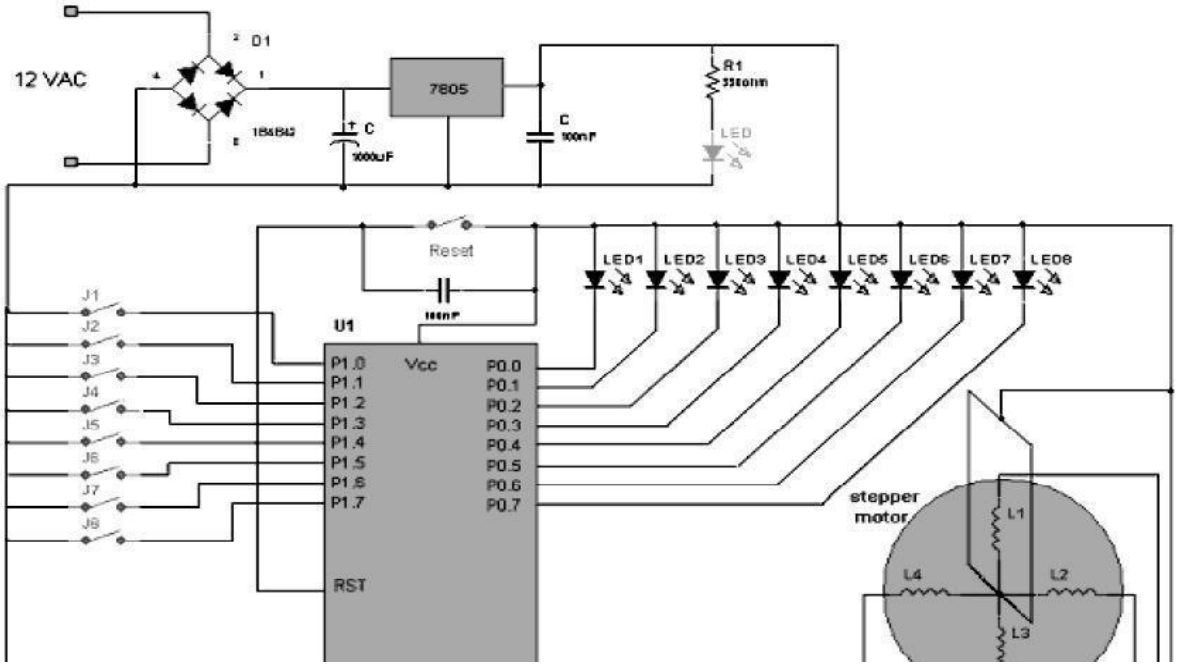


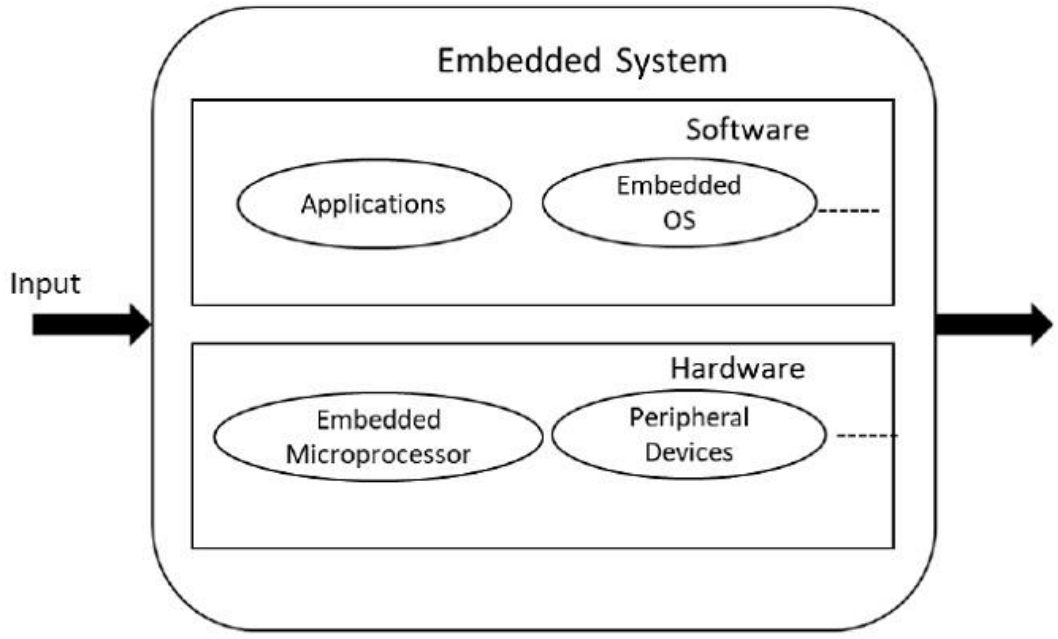
Fig.5.4 Stepper motor control board circuit

6. Embedded Operating System

What is an embedded operating system?

An embedded operating system is a specialized operating system (OS) designed to perform a specific task for a device that is not a computer.

- The main job of an embedded OS is to run the code that allows the device to do its job. The embedded OS also makes the device's hardware accessible to software that is running on top of the OS.
- An embedded OS often works within an embedded system.
- An embedded system is a computer that supports a machine.
- It performs one task in the bigger machine. Examples include computer systems in cars, traffic lights, digital televisions, ATMs, airplane controls, point of sale (POS) terminals, digital cameras, GPS navigation systems, elevators and Smart meters.
- Networks of devices containing embedded systems make up the internet of things (IoT). The embedded systems perform basic operations inside IoT devices, such as transferring data over a network without human interaction.



How does an embedded OS work?

An embedded OS enables an embedded device to do its job within a larger system. It communicates with the hardware of the embedded system to perform a specific function.

- For example, an elevator might contain an embedded system, such as a microprocessor or microcontroller, that lets it understand which buttons the

passenger is pressing. The embedded software that runs on that system is the embedded OS.

- In contrast to an OS for a general-purpose computer, an embedded OS has limited functionality. Depending on the device in question, the system may only run a single embedded application. However, that application is likely crucial to the device's operation. Given that, an embedded OS must be reliable and able to run with constraints on memory and processing power.
- In the case of a Raspberry PI system on a chip, an SD card acts as the device's hard drive and contains the code that runs on the device. The SD card is removable, so its contents can be modified on demand. Various operating systems can run on Raspberry PI devices. The embedded OS makes the device's hardware -- such as USB and HDMI ports -- accessible to the application running on top of the OS.

Examples of embedded OS devices

Some examples of devices with embedded OSes include the following:

- ATMs
- cellphones
- electric vehicles
- industrial control systems (ICS)
- Arduino-based devices

Arduino is an open source platform with a microcontroller that processes simple inputs, such as temperature or pressure, and turns them into outputs. These devices have a basic embedded OS that acts like a boot loader and a command interpreter.

An example of an Arduino-based device is a remote control car. The Arduino reads inputs from the car's controller and sends output information and commands to other components, such as the brakes.

Common uses of embedded OSes

Embedded OSes are put to a variety of uses, including the following:

- **ATMs.** ATMs have basic OSes that enable the machine to read a user's debit card and personal identification number input and perform bank account functions like withdrawal or checking balances. The OS does little else but react to user inputs and communicate with the ATM hardware.

- **Cellphones.** Cellphones require an OS like Android or iOS to boot the phone and enable applications to communicate with other phone hardware.
- **Electric vehicles.** Microcontrollers host embedded OSes that handle functions like braking or pressure sensing. For example, a certain amount of pressure on the front bumper may cause the airbag to go off. This type of function is known as reactive operation because it reacts to an input.
- **Industrial control systems.** Sensors are used in industrial control systems to measure factory conditions and send alerts if they become dangerous. Sensors contain an embedded OS that enable them to perform these tasks.
- **Traffic lights.** Embedded OSes enable a traffic light to cycle through different signals at programmed intervals.
- **Basic input/output system.** In some cases, BIOS could be considered an embedded OS because it is the firmware that enables a desktop computer's more complex OS to interact with the computer hardware.

Types of embedded OSes

Embedded OS are designed for the task they will perform. The various types of operating systems include the following:

- **Multitasking operating system.** A multitasking OS can perform several tasks at once. It uses job scheduling to perform basic tasks. For example, a cellphone OS divides up CPU resources among multiple tasks.
- **Real-time operating system.** A real-time OS is designed to be reactive. It processes inputs when they are received and responds within a specific timeframe. If the response time falls outside of the specified time period, the system could fail. Real-time OSes sometimes use rate monotonic scheduling, which assigns priorities to tasks.
- **Single loop control system.** This type of embedded OS exercises control over a single variable. An example would be temperature control in a smart home. A smart thermostat measures the temperature in the house and if it exceeds the limit set by the user, turns off the heat.

Advantages Embedded OS

The advantages of embedded operating system are as follows −

- Portable
- Much faster than other operating systems
- Less Hardware requirement

- Highly Predictable

Disadvantages

The disadvantages of embedded operating system are as follows −

- Less optimization
- High modification is required
- Customization is time taking process

Embedded vs. non-embedded OS: What's the difference?

- An embedded OS may reside on a chip within an electronic device. They are often limited in the scope of what they can do.
- In contrast, a non-embedded OS runs from a hard disk or a solid-state drive. Non-embedded OSes, such as Windows 10 or Mac OS, are configurable and upgradable. They are designed for general-purpose use.
- Another difference between embedded and non-embedded OSes is in how the operating system is coded. Embedded OSes are usually contained in a single executable image and execute one task. Desktop operating systems and network operating systems contain many applications.
- Embedded OSes also have a minimal or no user interface (UI). Embedded OSes, on the other hand, have a more dynamic UI.
- Embedded OSes and devices play a large part in the internet of things.

Multitasking Operating System

Multitasking refers to the simultaneous execution of multiple tasks (such as processes, programs, threads, and so on). Multi-tasking allows us to play a game, write code in a code editor, and surf the internet all at the same time on modern operating systems.

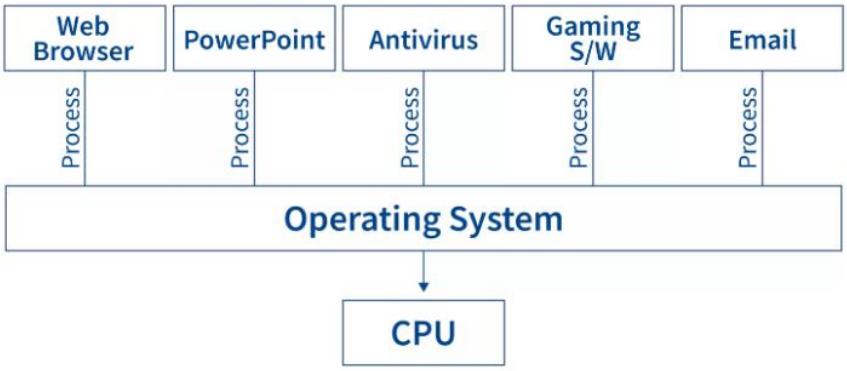
Why Multitasking Operating System?

Although multitasking was not completely enabled in early operating systems, they could not run many programs at once. As a result, a single piece of software might use the entire CPU of the computer to complete a task. The user was unable to do other operations, such as opening and shutting windows, because basic operating system capabilities, such as file copying, were disabled. Because modern operating systems provide comprehensive multitasking, multiple programs can run simultaneously without interfering with one another. Furthermore, many operating system processes can operate concurrently.

What is a Multitasking Operating System?

A multitasking operating system is a logical extension of multi-programming. Multitasking operating system varies from multiprogramming in that multi-programming relies simply on

the concept of context switching, whereas multitasking operating system incorporates time-sharing as well as context switching.



Types of Multitasking Operating Systems

Multitasking operating systems can be divided into three categories. The following are some examples:

1. Preemptive Multitasking
2. Cooperative Multitasking
3. True Multitasking

1. Preemptive Multitasking:

Preemptive multitasking is a task that a computer operating system is given. It determines how much time one job spends on the operating system before assigning another process to use it. The operating system is referred to as preemptive because it controls the entire operation.

Preemptive multitasking is used in desktop operating systems. Unix was the first operating system to implement this multitasking technique. The first versions of Windows to implement preemptive multitasking was Windows NT and Windows 95. In OS X, proactive multitasking was added to the Macintosh. This operating system notifies the programs when another application is ready to take over the CPU.

2. Cooperative Multitasking:

A cooperative multitasking operating system is referred to as 'Non-Preemptive Multitasking.' The basic goal of cooperative multitasking is to complete the current work while allowing another process to execute. taskYIELD() is used to complete this task. Context-switch is called when the taskYIELD() function is called.

Cooperative multitasking operating system was utilized by Windows and macOS. When a Windows program receives a message, It will do some quick work before handing over the CPU to the operating system until another message arrives. It worked perfectly as long as all of the programs were bug-free and designed with other programs in mind.

3. True Multitasking:

A true multitasking operating system is the ability to execute and process many tasks simultaneously without delay, rather than switching work from one processor to another. A true Multitasking operating system has the ability to conduct several tasks simultaneously while underlying the H/W or S/W.

Advantages of Multitasking Operating System

The following are some of the advantages of a multitasking operating system:

S. No.	Type	Description
1	Time Shareable	All jobs are given a defined amount of time, so they don't have to wait for the CPU.
2	Virtual Memory	Multitasking operating systems have the best virtual memory system. Any program does not require a long wait time to perform its tasks because of virtual memory; if this problem arises, those applications are shifted to virtual memory.
3	Secured Memory	Multitasking operating systems have well-defined memory management since they do not provide any types of permissions for undecided apps to consume memory.
4	Good Reliability	Multiple users are better satisfied with multitasking operating systems because they provide more flexibility. Every user can easily run single or numerous programs on a multitasking operating system.
5	Manage Several Users	This operating system is better adapted to support several users at once, and multiple apps can run simultaneously without slowing down the system.
6	Optimize Computer Resources	A multitasking operating system may manage the resources of several computers, including RAM, input/output devices, CPU, hard disc, and so on.
7	Background Processing	Background processes can operate more efficiently under a multitasking operating system. Most users are unaware of these background processes, although they aid the smooth operation of other programs such as firewalls, antivirus software, and others.
8	Use Multiple Programs	Users can run multiple programs at once, including an internet browser, a code editor, Microsoft Word, Microsoft Excel, PowerPoint, games, and other utilities.

Disadvantages of Multitasking Operating System

The following are some of the disadvantages of multitasking operating system:

S. No.	Type	Description
1	Processor Boundation	Because of the poor speed of its processors, the system may run applications slowly, and their reaction time may increase when processing many programs. More computing power is necessary to solve the processing foundation challenge.
2	CPU Heat up	In a multitasking operating system, the CPU generates more heat since multiple processors are engaged at the same time to execute any task.
3	Memory Boundation	The computer's performance may suffer as a result of many programs running at the same time, as the main memory becomes overloaded when multiple programs are loaded. Reaction time grows since the CPU is unable to give distinct times for each program. The usage of low-capacity RAM is the primary cause of this problem. As a result, the RAM capacity can be increased to meet the requirements.

Examples of Multitasking Operating Systems

Some examples of multitasking operating systems are as follows:

- **Windows XP:** Microsoft's Windows NT operating system has been updated with Windows XP. For professional users, it is the immediate successor to Windows 2000, whereas, for residential users, it is Windows Me.
- **Windows Vista:** Microsoft's Windows Vista is a version of the Windows NT operating system. It was the direct successor to Windows XP, which had been launched five years prior, marking the longest period between successive Microsoft Windows desktop operating system releases at the time.
- **Windows 7:** Windows 7 is the latest version of Microsoft's Windows NT operating system. On July 22, 2009, it was launched to manufacture, and on October 22, 2009, it was made commercially available. It succeeds Windows Vista, which was introduced approximately three years ago.
- **Windows 8:** Windows 8 is the latest version of Microsoft's Windows NT operating system.
- **Windows 10:** Microsoft's Windows NT operating system has been updated with Windows 10. It's the immediate successor to Windows 8.1, which came out over two years ago.
- **Windows 2000:** Microsoft's Windows 2000 is a major release of the Windows NT operating system geared toward enterprises. It was Windows NT 4.0's direct successor.
- **IBM’s OS/390:** The IBM operating system for System/390 mainframe systems is OS/390.

- **UNIX:** Unix is a multitasking, multiuser computer operating system derived from the original AT&T Unix, which was developed by Ken Thompson, Dennis Ritchie, and others at the Bell Labs research center in 1969.

Conclusion

- Multitasking is the process of performing multiple tasks simultaneously.
- In a multitasking operating system, the CPU switches between many tasks to complete them.
- Isolation and memory protection is provided by multitasking operating systems.
- A multitasking operating system utilizes the concept of many processors.
- A multitasking operating system can run multiple tasks at the same time.
- In a multitasking operating system, context-switching and time-sharing are used.
- CPU idle time is significantly decreased in the multitasking operating system.
- 1 CPU is required in a multitasking operating system.

Real-Time Operating System (RTOS)

Real-time operating system (RTOS) is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay.

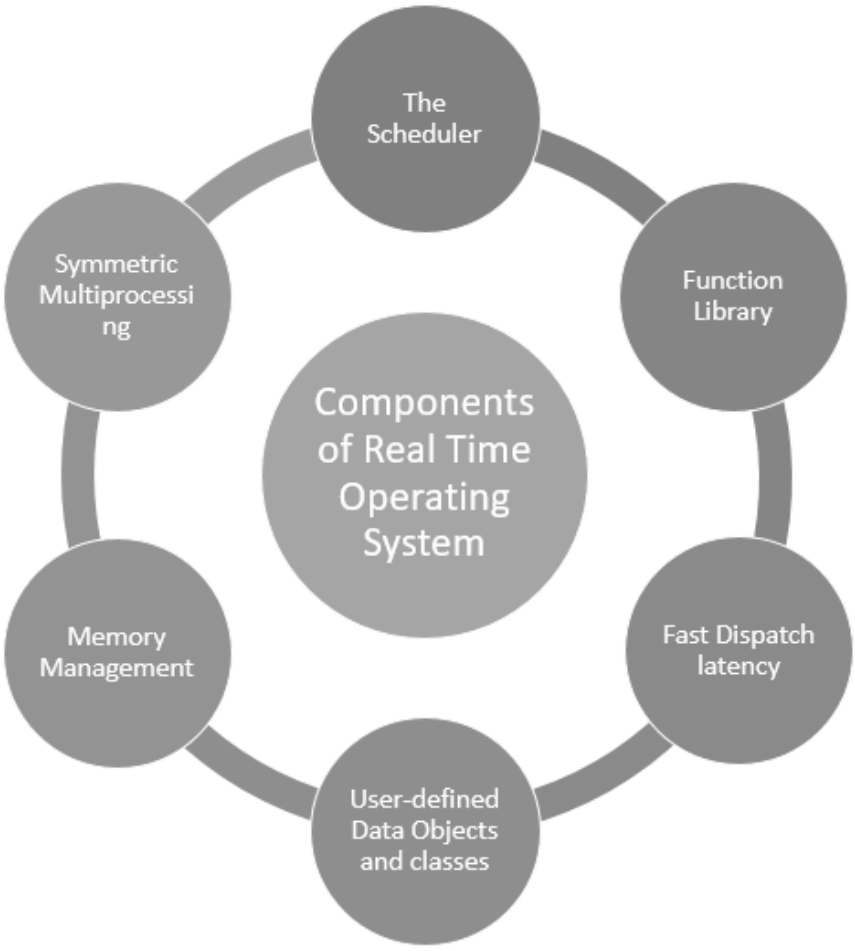
- In a RTOS, Processing time requirement are calculated in tenths of seconds increments of time.
- It is time-bound system that can be defined as fixed time constraints.
- In this type of system, processing must be done inside the specified constraints. Otherwise, the system will fail.

Why use an RTOS?

Here are important reasons for using RTOS:

- It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing.
- The Real time OS provides API functions that allow cleaner and smaller application code.
- Abstracting timing dependencies and the task-based design results in fewer interdependencies between modules.
- RTOS offers modular task-based development, which allows modular task-based testing.
- The task-based API encourages modular development as a task, will typically have a clearly defined role. It allows designers/teams to work independently on their parts of the project.
- An RTOS is event-driven with no time wastage on processing time for the event which is not occur

Components of RTOS



Components of Real Time Operating System

Here, are important Component of RTOS

(i) **The Scheduler**: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.

(ii) **Symmetric Multiprocessing (SMP)**: It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.

(iii) **Function Library**: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

(iv) **Memory Management**: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.

(v) **Fast dispatch latency**: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue, that has started processing.

(vi) **User-defined data objects and classes**: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.

Types of RTOS

Three types of RTOS systems are:

1. Hard Real Time

In Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration.

Example: Medical critical care system, Aircraft systems, etc.

2. Firm Real time

These type of RTOS also need to follow the deadlines. However, missing a deadline may not have big impact but could cause undesired affects, like a huge reduction in quality of a product.

Example: Various types of Multimedia applications.

3. Soft Real Time

Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.

Example: Online Transaction system and Livestock price quotation System.

Terms used in RTOS

Here, are essential terms used in RTOS:

- **Task –** A set of related tasks that are jointly able to provide some system functionality.
- **Job –** A job is a small piece of work that can be assigned to a processor, and that may or may not require resources.
- **Release time of a job –** It's a time of a job at which job becomes ready for execution.
- **Execution time of a job:** It is time taken by job to finish its execution.
- **Deadline of a job:** It's time by which a job should finish its execution.
- **Processors:** They are also known as active resources. They are important for the execution of a job.
- **Maximum It is the** allowable response time of a job is called its relative deadline.
- **Response time of a job:** It is a length of time from the release time of a job when the instant finishes.
- **Absolute deadline:** This is the relative deadline, which also includes its release time.

Features of RTOS

Here are important features of RTOS:

- Occupy very less memory
- Consume fewer resources
- Response times are highly predictable
- Unpredictable environment

- The Kernel saves the state of the interrupted task ad then determines which task it should run next.
- The Kernel restores the state of the task and passes control of the CPU for that task.

Factors for selecting an RTOS

Here, are essential factors that you need to consider for selecting RTOS:

- **Performance**: Performance is the most important factor required to be considered while selecting for a RTOS.
- **Middleware**: if there is no middleware support in Real time operating system, then the issue of time-taken integration of processes occurs.
- **Error-free**: RTOS systems are error-free. Therefore, there is no chance of getting an error while performing the task.
- **Embedded system usage**: Programs of RTOS are of small size. So we widely use RTOS for embedded systems.
- **Maximum Consumption**: we can achieve maximum Consumption with the help of RTOS.
- **Task shifting**: Shifting time of the tasks is very less.
- **Unique features**: A good RTS should be capable, and it has some extra features like how it operates to execute a command, efficient protection of the memory of the system, etc.
- **24/7 performance**: RTOS is ideal for those applications which require to run 24/7.

Difference between in GPOS and RTOS

Here are important differences between GPOS and RTOS:

General-Purpose Operating System (GPOS)	Real-Time Operating System (RTOS)
It used for desktop PC and laptop.	It is only applied to the embedded application.
Process-based Scheduling.	Time-based scheduling used like round-robin scheduling.
Interrupt latency is not considered as important as in RTOS.	Interrupt lag is minimal, which is measured in a few microseconds.
No priority inversion mechanism is present in the system.	The priority inversion mechanism is current. So it cannot modify by the system.
Kernel's operation may or may not be preempted.	Kernel's operation can be preempted.
Priority inversion remain unnoticed	No predictability guarantees